EECS 2500 – Linear Data Structures – Fall 2021
Programming Lab #1: Due Monday, 09/27/2021 @ 11:59PM

This project is designed to help give students practice with stacks.

Assume you have a set of cards laid out in an *n* by *m* grid (*n* rows of *m* columns each). Some are face up, and others are face down. We can collapse the grid into a single pile by using a series of flips, each of which is one of the four following types:

Top Flip: Here the cards in the top row are flipped over onto the corresponding cards on the row beneath them. Note that if a card is face up in the top row, it becomes face down after the flip, and vice versa. If the top row contains one or more piles of cards, each entire pile is flipped over like a stack of pancakes as it is moved to the lower row. Note: Conceptually, we think of flipping the ENTIRE stack at once; what actually happens is that each card in the stack gets flipped one at a time, until that stack has no cards left.

Bottom Flip: Same as the Top Flip, except the bottom row is flipped onto the next-to-bottom row.

Left Flip: Flip the cards in the left-most column onto the next-to-leftmost column.

Right Flip: Flip the cards in the rightmost column onto the next-to-rightmost column.

After a series of $n + m - 2$ flips, the cards will be in a single pile, some cards face up and some face down. Your job is to determine the order of the face-down cards in this final pile.

Input:

Each test case will start with a line containing two positive integers *n m* indicating the number of rows and columns in the grid. There will be a space between *n* and *m*.

After this will come *n* rows of *m* strings indicating each card's number and its orientation. (the first row is the top row and the first value in each row is the leftmost card.)

Each string will be of the format OSV, where:

O is the orientation: either 'U' for "face-Up", or 'F' for "face-Down"
S is the suit: one of 'C', 'D', 'H', 'S' for "Clubs", "Diamonds", "Hearts", or "Spades", respectively
V is the face Value: one of '2', ... '9', 'T' (ten), 'J' (Jack), 'Q' (queen), 'K' (King), or 'A' (Ace)

Thus, "USA" indicates "Ace of spaces, face-up", and "DC3" represents "3 of Clubs, face-down"

You will have to (A) make sure that each OSV string is valid, and then (B) convert each to an integer in the range of -52 to +52, representing the four suits and 13 values, with the sign representing the orientation. Zero will not be used. If a string is not valid, your program must end. Immediately, with no error message (a rather user-unfriendly and heavy-handed way of handling errors, but this program isn't necessarily about the input's validity so much as it is about processing the stacks. Your input must accept upper- or lower-case characters (card identification strings are not case-sensitive).

After these *n* rows there will be one more line containing $n + m - 2$ characters indicating the order in which to apply the flips to the grid. Each character will be either 'T', 'B', 'L' or 'R' corresponding to a top, bottom, left or right flip. All flip sequences will be legal, i.e., you won't be asked to do more than $n - 1$ top and bottom flips or $m - 1$ left and right flips. You are not responsible for making sure that the flip directions are one of TBLR – I won't test your code with an illegal flip direction.

The maximum value for *n* and *m* is 20. A line containing two zeros will terminate input.

Obviously, if $m * n$ is > 52, it will require more than one deck of cards. Repeated cards should be a non-issue for your program.

Output:
For each test, output the word "Test", a single space, the test number, a colon, and a space, followed by a list of the SUITS AND VALUES of all of the face-down cards in the final deck, starting from the bottom of the deck. Follow the format used in the examples *exactly*.

Sample Input:

```
2 3
UC4 DD3 DC8
UC6 UDJ DC5
LRB
1 1
UC3

1 1
DC3

0 0
```

Sample Output

```
Test 1: D3 C4 C5 DJ
Test 2:
Test 3: C3
```

Let's look closely at the first example -- the starting point is:

4 of Clubs (up)	3 of Diamonds (down)	8 of Clubs (down)
6 of Clubs (up)	J of Diamonds (up)	5 of clubs (down)

The first flip (L), puts the 4 of clubs ON TOP OF the 3 of diamonds, and flips it from face-up to down. It also flips the 6 of clubs ON TOP of the J of diamonds, and flips it from face-up to face-down.

Now we have (each stack is listed top-to-bottom):

[4 C (down), 3 D (down)]	8 C (down)
[6 C (down), J D (up)]	5 C (down)

The second flip (R), flips the 8 of clubs from face-down to face-up onto top of the stack to its left and the same for the 5 of clubs. The new configuration leaves us with two stacks:

[8 C (up), 4 C (down), 3 D (down)]
[5 C (up), 6 C (down), J D (up)]

The final flip, B, flips the bottom stack on top of the first
[J D (down), 6 C (up), 5 C (down), 8 C (up), 4 C (down), 3 D (down)]

Reading this stack from bottom to top, looking for the face-down cards, we have:

3 of Diamonds, 4 of Clubs, 5 of Clubs, and the Jack of Diamonds, giving us the output:

```
D3 C4 C5 DJ
```

In this example, the flip sequence was LRB. With the same cards, a sequence of `LLT` would result in
`DJ 3D 4C`

You are encouraged to get a deck of cards and work through this example yourself.

You must implement your stack as a linked list of nodes. Your stack mut be generic (<T>), and you must use it with `Integer` as its type.

You ***must*** put `main()` in a class called `CardStack`.

Your program should allow a user to input multiple cases of *n* and *m*, card numbers, and commands, create the grid, and then produce the output as shown above.

THERE MUST BE NO TEXT PROMPT TO THE CONSOLE IN YOUR FINAL SUBMISSION

You should test your program on a variety of scenarios, and make absolutely sure that it handles them all.

Submission Instructions:

MAKE A NEW WORKSPACE FOR THIS COURSE – DO NOT RE-USE YOUR 1510 WORKSPACE!

Submit a 7-zip file of your ENTIRE Java Workspace directory (and its subfolders) to Blackboard (don't submit just the project folder *within* the workspace; submit the whole workspace). I should be able to unzip your workspace, point Eclipse to it, and run your code without modification.

If you submit less than an entire workspace, or if your code has syntax errors and will not compile, it will receive zero credit.

Your *project* MUST be named "2500-Lab1", and the class containing main MUST be named **`CardStack.`**

Your 7-zip file MUST be named "2500-<LastName><comma><space><FirstName>.7z (example: "2500-Smith, John.7z".

If you have questions, please let me know ASAP.

The standard Academic Integrity policies apply – your code is to be yours and yours alone. You are not to use ANY resources other than the text, lecture material, or those given above in the assignment (this includes any online sources). Do not share your code, or even your approach to solving the problem, with each other. You should not use (or need to use) anything from the Java library (i.e., things to import) other than java.util.Scanner.

If you come across something else you THINK you need to `import` and use, check with me first. Things like collections, sets, lists, and linkedlists are categorically not allowed – this is an exercise where you have to write all of the tools.